題目不是分類問題，輸出的變數並不是類別變數，而是1個實數，請記得output數量設定1 另外，迴歸問題通常是拿MSE來當訓練模型的loss，請不要拿類別用的CategoricalCrossentropy 評估模型也不會用到 遇到分類問題，請參考作業6、7、8

兩題的結論都在最下方，這邊建立了Ridge模型與keras模型進行比較

In [1]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
from sklearn.datasets import load_diabetes
X, y = load_diabetes(return_X_y=True,as_frame=True)
```

In [3]:
```
X # 待會會把sex轉onehot並取代
```

Out[3]:

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019908	-0.017646
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068330	-0.092204
2	0.085299	0.050680	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	-0.002592	0.002864	-0.025930
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022692	-0.009362
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031991	-0.046641
...	...	...	...	...	...	...	...	...	...	...
437	0.041708	0.050680	0.019662	0.059744	-0.005697	-0.002566	-0.028674	-0.002592	0.031193	0.007207
438	-0.005515	0.050680	-0.015906	-0.067642	0.049341	0.079165	-0.028674	0.034309	-0.018118	0.044485
439	0.041708	0.050680	-0.015906	0.017282	-0.037344	-0.013840	-0.024993	-0.011080	-0.046879	0.015491
440	-0.045472	-0.044642	0.039062	0.001215	0.016318	0.015283	-0.028674	0.026560	0.044528	-0.025930
441	-0.045472	-0.044642	-0.073030	-0.081414	0.083740	0.027809	0.173816	-0.039493	-0.004220	0.003064

442 rows × 10 columns

In [4]:
```
import pandas as pd
# 要儲存各種模型的 metrics(見最後結果)
methods = ["_ridge", "_ridge_nbp", "_API", "_API_nbp"]
metric_list = ["mse", "r2"]
for method in methods:
    for metric in metric_list:
        locals()[metric+method]=[]
loss_API = []
loss_API_nbp = []
# 存結果的位置
results = pd.DataFrame({'mean_mse':[],'mean_r2':[]}).T
```

In [5]:
```
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder()
sex = enc.fit_transform(X[['sex']]).toarray()

# 先drop掉sex 然後加上onehot後的sex
X = X.drop(['sex'],axis=1)
X[['sex1','sex2']] = sex
```

In [6]:
```
X #已經將sex 轉 onehot
```

Out[6]:

	age	bmi	bp	s1	s2	s3	s4	s5	s6	sex1	sex2
0	0.038076	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019908	-0.017646	0.0	1.0
1	-0.001882	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068330	-0.092204	1.0	0.0
2	0.085299	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	-0.002592	0.002864	-0.025930	0.0	1.0
3	-0.089063	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022692	-0.009362	1.0	0.0
4	0.005383	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031991	-0.046641	1.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...
437	0.041708	0.019662	0.059744	-0.005697	-0.002566	-0.028674	-0.002592	0.031193	0.007207	0.0	1.0
438	-0.005515	-0.015906	-0.067642	0.049341	0.079165	-0.028674	0.034309	-0.018118	0.044485	0.0	1.0
439	0.041708	-0.015906	0.017282	-0.037344	-0.013840	-0.024993	-0.011080	-0.046879	0.015491	0.0	1.0
440	-0.045472	0.039062	0.001215	0.016318	0.015283	-0.028674	0.026560	0.044528	-0.025930	1.0	0.0
441	-0.045472	-0.073030	-0.081414	0.083740	0.027809	0.173816	-0.039493	-0.004220	0.003064	1.0	0.0

442 rows × 11 columns

對整理好的data分成train與test

In [7]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

方法1: Ridge Regression

In [8]:
```
from sklearn.linear_model import Ridge
ridge = Ridge(alpha=0.01)
```

方法2: Functional API

In [9]:
```
from tensorflow import keras
from tensorflow.keras import layers
```

In [10]:
```
# 設置 keras model : (有包含bp)

n, p = X.shape
# Input 1 (放置5個變數['age', 'sex1', 'sex2', 'bmi', 'bp'])
x_1 = keras.Input(shape=(5,), name='x_1')
h1_1 = layers.Dense(16,
                    activation='relu',
                    name='h1_1')(x_1)
h2_1 = layers.Dense(32,
                    activation='relu',
                    name='h2_1')(h1_1)
o_1 = layers.Dense(8, name = 'o_1')(h2_1)
# Input 2 (放置3個變數['s1', 's2', 's3'])
x_2 = keras.Input(shape=(3,), name='x_2')
h1_2 = layers.Dense(16,
                    activation='relu',
                    name='h1_2')(x_2)
h2_2 = layers.Dense(32,
                    activation='relu',
                    name='h2_2')(h1_2)
o_2 = layers.Dense(8, name = 'o_2')(h2_2)
# Input 3 (放置3個變數['s4', 's5', 's6'])
x_3 = keras.Input(shape=(3,), name='x_3')
h1_3 = layers.Dense(16,
                    activation='relu',
                    name='h1_3')(x_3)
h2_3 = layers.Dense(32,
                    activation='relu',
                    name='h2_3')(h1_3)
o_3 = layers.Dense(8, name = 'o_3')(h2_3)
h3 = layers.Concatenate(name='h3')([o_1, o_2, o_3]) #這邊使用串聯
# h3 = layers.Add(name='h3')([o_1, o_2, o_3])       #你也可以相加
outputs = layers.Dense(1,
                       activation='linear',
                       name = 'y')(h3)

model = keras.Model(inputs=[x_1,x_2,x_3], outputs=outputs)
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
x_1 (InputLayer)                [(None, 5)]          0
__________________________________________________________________________________________________
x_2 (InputLayer)                [(None, 3)]          0
__________________________________________________________________________________________________
x_3 (InputLayer)                [(None, 3)]          0
__________________________________________________________________________________________________
h1_1 (Dense)                    (None, 16)           96          x_1[0][0]
__________________________________________________________________________________________________
h1_2 (Dense)                    (None, 16)           64          x_2[0][0]
__________________________________________________________________________________________________
h1_3 (Dense)                    (None, 16)           64          x_3[0][0]
__________________________________________________________________________________________________
h2_1 (Dense)                    (None, 32)           544         h1_1[0][0]
__________________________________________________________________________________________________
h2_2 (Dense)                    (None, 32)           544         h1_2[0][0]
__________________________________________________________________________________________________
h2_3 (Dense)                    (None, 32)           544         h1_3[0][0]
__________________________________________________________________________________________________
o_1 (Dense)                     (None, 8)            264         h2_1[0][0]
__________________________________________________________________________________________________
o_2 (Dense)                     (None, 8)            264         h2_2[0][0]
__________________________________________________________________________________________________
o_3 (Dense)                     (None, 8)            264         h2_3[0][0]
__________________________________________________________________________________________________
h3 (Concatenate)                (None, 24)           0           o_1[0][0]
                                                                 o_2[0][0]
                                                                 o_3[0][0]
__________________________________________________________________________________________________
y (Dense)                       (None, 1)            25          h3[0][0]
==================================================================================================
Total params: 2,673
Trainable params: 2,673
Non-trainable params: 0
__________________________________________________________________________________________________
```

In [11]:
```
# 因為畫流程圖套件複雜，此圖僅供理解用
# from tensorflow.keras.utils import plot_model
# plot_model(model)
```

In [12]:
```
model.compile(optimizer='adam',
                  loss=keras.losses.MeanSquaredError())
```

In [13]:
```
# 設置 keras model_nbp : (不包含bp) #注意input變數的設定

n, p = X.shape
# Input 1 (放置4個變數['age', 'sex1', 'sex2', 'bmi'])
x_1 = keras.Input(shape=(4,), name='x_1')
h1_1 = layers.Dense(16,
                    activation='relu',
                    name='h1_1')(x_1)
h2_1 = layers.Dense(32,
                    activation='relu',
                    name='h2_1')(h1_1)
o_1 = layers.Dense(8, name = 'o_1')(h2_1)
# Input 2 (放置3個變數['s1', 's2', 's3'])
x_2 = keras.Input(shape=(3,), name='x_2')
h1_2 = layers.Dense(16,
                    activation='relu',
                    name='h1_2')(x_2)
h2_2 = layers.Dense(32,
                    activation='relu',
                    name='h2_2')(h1_2)
o_2 = layers.Dense(8, name = 'o_2')(h2_2)
# Input 3 (放置3個變數['s4', 's5', 's6'])
x_3 = keras.Input(shape=(3,), name='x_3')
h1_3 = layers.Dense(16,
                    activation='relu',
                    name='h1_3')(x_3)
h2_3 = layers.Dense(32,
                    activation='relu',
                    name='h2_3')(h1_3)
o_3 = layers.Dense(8, name = 'o_3')(h2_3)
h3 = layers.Concatenate(name='h3')([o_1, o_2, o_3]) #這邊使用串聯
# h3 = layers.Add(name='h3')([o_1, o_2, o_3])       #你也可以相加
outputs = layers.Dense(1,
                       activation='linear',
                       name = 'y')(h3)

model_nbp = keras.Model(inputs=[x_1,x_2,x_3], outputs=outputs)
model_nbp.summary()
```

```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
x_1 (InputLayer)                [(None, 4)]          0
__________________________________________________________________________________________________
x_2 (InputLayer)                [(None, 3)]          0
__________________________________________________________________________________________________
x_3 (InputLayer)                [(None, 3)]          0
__________________________________________________________________________________________________
h1_1 (Dense)                    (None, 16)           80          x_1[0][0]
__________________________________________________________________________________________________
h1_2 (Dense)                    (None, 16)           64          x_2[0][0]
__________________________________________________________________________________________________
h1_3 (Dense)                    (None, 16)           64          x_3[0][0]
__________________________________________________________________________________________________
h2_1 (Dense)                    (None, 32)           544         h1_1[0][0]
__________________________________________________________________________________________________
h2_2 (Dense)                    (None, 32)           544         h1_2[0][0]
__________________________________________________________________________________________________
h2_3 (Dense)                    (None, 32)           544         h1_3[0][0]
__________________________________________________________________________________________________
o_1 (Dense)                     (None, 8)            264         h2_1[0][0]
__________________________________________________________________________________________________
o_2 (Dense)                     (None, 8)            264         h2_2[0][0]
__________________________________________________________________________________________________
o_3 (Dense)                     (None, 8)            264         h2_3[0][0]
__________________________________________________________________________________________________
h3 (Concatenate)                (None, 24)           0           o_1[0][0]
                                                                 o_2[0][0]
                                                                 o_3[0][0]
__________________________________________________________________________________________________
y (Dense)                       (None, 1)            25          h3[0][0]
==================================================================================================
Total params: 2,657
Trainable params: 2,657
Non-trainable params: 0
__________________________________________________________________________________________________
```

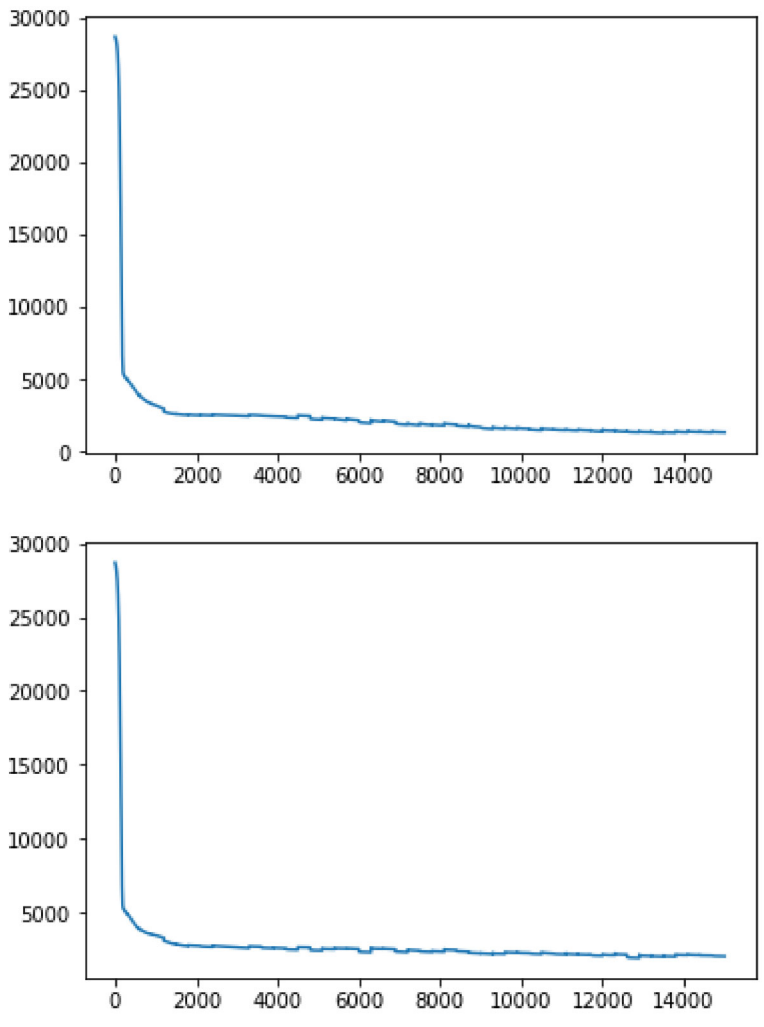
In [14]:
```
model_nbp.compile(optimizer='adam',
                  loss=keras.losses.MeanSquaredError())
```

In [15]:
```
# 利用 RepeatedKFold 驗證
from sklearn.model_selection import RepeatedKFold
from sklearn.metrics import mean_squared_error, r2_score
```

In [16]:
```
# repeat 10-Fold 5 次
rkf = RepeatedKFold(n_splits=10,n_repeats=5,random_state=100)
for train_index, test_index in rkf.split(X, y):
    X_train, X_test = X.loc[train_index,:], X.loc[test_index,:]
    y_train, y_test = y[train_index], y[test_index]


    ### 考慮血壓
    # Fit ridge model
    rid = ridge.fit(X_train, y_train)

    # Fit keras API model
    n = X_train.shape[0]
    history = model.fit({'x_1': X_train[['age', 'sex1', 'sex2', 'bmi', 'bp']],
                         'x_2': X_train[['s1', 's2', 's3']],
                         'x_3': X_train[['s4', 's5', 's6']]},
                         {'y': y_train},
                         batch_size=n,
                         epochs=300,
                         verbose=0)
    # 把loss存起來
    loss_API.extend(history.history['loss'])

    # predict
    mse_ridge.append(mean_squared_error(y_test, rid.predict(X_test)))
    r2_ridge.append(r2_score(y_test, rid.predict(X_test)))
    ypred = model.predict({'x_1': X_test[['age', 'sex1', 'sex2', 'bmi', 'bp']],
                           'x_2': X_test[['s1', 's2', 's3']],
                           'x_3': X_test[['s4', 's5', 's6']]})
mse_API.append(mean_squared_error(y_test,ypred))
r2_API.append(r2_score(y_test,ypred))




### 不考慮血壓
# Fit ridge model
rid = ridge.fit(X_train.drop(['bp'],axis=1), y_train)

# Fit model_nbp (這裡考慮沒有血壓變數 bp)
n = X_train.shape[0]
history = model_nbp.fit({'x_1': X_train[['age', 'sex1', 'sex2', 'bmi']],
                         'x_2': X_train[['s1', 's2', 's3']],
                         'x_3': X_train[['s4', 's5', 's6']]},
                        {'y': y_train},
                        batch_size=n,
                        epochs=300,
                        verbose=0)
# 把loss存起來
loss_API_nbp.extend(history.history['loss'])

# predict
mse_ridge_nbp.append(mean_squared_error(y_test, rid.predict(X_test.drop(['bp'],axis=1)))) #記得drop
r2_ridge_nbp.append(r2_score(y_test, rid.predict(X_test.drop(['bp'],axis=1))))           #記得drop
ypred = model_nbp.predict({'x_1': X_test[['age', 'sex1', 'sex2', 'bmi']],
                           'x_2': X_test[['s1', 's2', 's3']],
                           'x_3': X_test[['s4', 's5', 's6']]})
mse_API_nbp.append(mean_squared_error(y_test,ypred))
r2_API_nbp.append(r2_score(y_test,ypred))
```

In [17]:
```
# 畫keras API的loss
print('model_tf 的 loss:')
plt.plot(loss_API)
plt.show()
plt.plot(loss_API_nbp)
plt.show()
```

```
model_tf 的 loss:
```



In [18]:
```
# 存下來
results['Ridge']=[np.mean(mse_ridge),np.mean(r2_ridge)]
results['Ridge_nbp']=[np.mean(mse_ridge_nbp),np.mean(r2_ridge_nbp)]
results['Functional_API']=[np.mean(mse_API),np.mean(r2_API)]
results['Functional_API_nbp']=[np.mean(mse_API_nbp),np.mean(r2_API_nbp)]
```

結果

In [19]:
```
results
```

Out[19]:

	Ridge	Ridge_nbp	Functional_API	Functional_API_nbp
mean_mse	3010.521048	3162.961026	2551.576470	2852.462732
mean_r2	0.471427	0.442727	0.550108	0.493102

1.比較Ridge與Functional API的結果可以知道，我使用Functional API所訓練的結果較優(相同data下，mse較小)，這是我目前創建較好的模型 (以示範code為主，還請同學自行尋找更好的模型)

2.兩個模型同時拿掉bp這個變數，可以發現R^2都有變小的趨勢(解釋能力)，其mse也變大了，這表示bp這個變數在這個模型具有相當的影響力

In [ ]:
```
```